CLOUDS: A Decision Tree Classifier for Large Datasets

Khaled Alsabti
Department of EECS
Syracuse University

Sanjay Ranka
Department of CISE
University of Florida

Vineet Singh
Information Technology Lab
Hitachi America, Ltd.

Abstract

Classification for very large datasets has many practical applications in data mining. Techniques such as discretization and dataset sampling can be used to scale up decision tree classifiers to large datasets. Unfortunately, both of these techniques can cause a significant loss in accuracy. We present a novel decision tree classifier called CLOUDS, which *samples* the splitting points for numeric attributes followed by an *estimation* step to narrow the search space of the best split. CLOUDS reduces computation and I/O complexity substantially compared to state of the art classifiers, while maintaining the quality of the generated trees in terms of accuracy and tree size. We provide experimental results with a number of real and synthetic datasets.

Introduction

Classification is the process of generating a description or a model for each class of a given dataset. The dataset consists of a number of records, each consisting of several fields called *attributes* that can either be *categorical* or *numeric*. Each record belongs to one of the given (categorical) classes. The set of records available for developing classification methods are generally decomposed into two disjoint subsets, training set and test set. The former is used for deriving the classifier, while the latter is used to measure the accuracy of the classifier. Decision-tree classifiers SLIQ and SPRINT have been shown to achieve good accuracy, compactness and efficiency for very large datasets (Agrawal, Mehta, & Rissanen 1996; Agrawal, Mehta, & Shafer 1996); the latter has substantially superior computational characteristics for large datasets. Deriving a typical decision tree classifier from the training set consists of two phases: construction and pruning. Since the construction phase is the computationally more intensive portion (Agrawal, Mehta, & Shafer 1996), the main focus of this paper is the construction phase.

CART, SLIQ, and SPRINT use the *gini* index to derive the splitting criterion at every internal node of the tree. The use of the *gini* index is computationally challenging. Efficient methods such as SPRINT require sorting along each of the numeric attributes, which requires the use of memory-resident hash tables to split the sorted attribute lists at every level. For datasets that are significantly larger than the available memory, this will require multiple passes over the entire dataset. In this paper we present a simple scheme that eliminates the need to sort along each of the numeric attributes by exploiting the following properties of the *gini* index for a number of real datasets: (1) The value of the *gini* index along a given attribute generally increases or decreases slowly. The number of good local minima is small compared to the size of the entire dataset. This is especially true for attributes along which the best splitting point is obtained. (2) The minimum value of the *gini* index for the splitting point along the splitting attribute is (relatively) significantly lower than most of the other points along the splitting as well as other attributes.

We show that the above properties can be used to develop an I/O and computationally efficient method for estimating the split at every internal node. Experimental results on real and synthetic datasets using our method show that, in most cases, the splitting point derived has the same *gini* index as the one obtained by sorting along each of the attributes. Using this new method for splitting at every internal node, we have developed a new decision tree classifier called CLOUDS (Classification for Large or OUt-of-core DataSets). CLOUDS is shown to have substantially lower computation and I/O requirements as compared to SPRINT for a number of real and synthetic datasets. The accuracy and compactness of the decision trees generated is the same or comparable to CART, C4, and SPRINT.

Estimating the Splitting Point for Numeric Attributes

In this section we present two new methods for estimating the splitting point for the numeric attributes. These methods are used by CLOUDS at each node of the tree. We compare the new methods with a direct method and a simple sampling method. The *Direct Method* (*DM*)

sorts the values of each numeric attribute, and evaluates the *gini* index at each possible split point to find the splitting point with the minimum *gini* index. The sorting operation requires $O(n \lg n)$ time, where n is the number of values. It requires several passes of I/O for disk resident-data. The *Dataset Sampling* (*DS*) method draws a random subset of the whole dataset. The *DM* method is applied to this sample to build the classifier. In order to study the quality of this method, the *gini* index of the splitter (the point used for partitioning the data) is obtained using the whole dataset.

In this paper we present two new methods based on *Sampling the Splitting Points*, i.e., sampling the computation. The first method, Sampling the Splitting points (SS), derives the splitter from a limited number of the splitting points. The second method, *SSE* (Sampling the Splitting points with Estimation), improves upon *SS* by estimating the *gini* values to narrow the search space of the best split, and it is designed to derive the best or near-best splitter with respect to the *gini* index. Both the methods evaluate the *gini* index at only a subset of splitting points along each numeric attribute.

Sampling the Splitting points (SS) In the *SS* method, the range of each numeric attribute is divided into q intervals using a quantiling-based technique. Each interval contains approximately the same number of points. For each numeric attribute, the *gini* index is evaluated at the interval boundaries. This is used to determine the minimum *gini* value ($gini_{min}$) among all interval boundaries of the numeric attributes and across all the categorical attributes. The split point with $gini_{min}$ will be used as a splitter. The *SS* method requires one pass over the dataset to derive the splitting point.

Sampling the Splitting points with Estimation (SSE) The *SSE* method divides the range of each numeric attribute and finds the minimum *gini* value ($gini_{min}$), as in the *SS* method. Further, it estimates a lower bound $gini^{est}$ of the *gini* value for each interval. All the intervals with $gini^{est} \geq gini_{min}$ are eliminated (prune) to derive a list of potential candidate intervals (***alive*** intervals). For an ***alive*** interval, we evaluate the *gini* index at every distinct point in the interval to determine the splitter. This may require another pass over the entire dataset. The *SSE* method estimates the minimum value of the *gini* index in a given interval $[v_l, v_u]$ using the statistics of the interval endpoints. In the following we describe our estimation heuristic. Let n and c be the number of records and the number of classes, respectively. We use the following notation:

- x_i (y_i) — the number of elements of class i that are less than or equal to v_l (v_u)
- c_i — the total number of elements of class i
- n_l (n_u) — the number of elements that are less than or equal to v_l (v_u)

The $gini^D$ at point v_l of a numeric attribute a for dataset S is given by:

$$gini^D(S, a \leq v_l) = \frac{n_l}{n}(1 - \sum_{i=1}^{c}(\frac{x_i}{n_l})^2) + \frac{n - n_l}{n}(1 - \sum_{i=1}^{c}(\frac{c_i - x_i}{n - n_l})^2) \quad (1)$$

We use a hill-climbing algorithm to estimate the lower bound of the *gini* index in the given interval. This algorithm makes a series of decisions to estimate the lower bound, with each decision based on the values of *gradient* along a subset of the c classes. Gradients along a subset of these c classes are calculated and the direction along the minimum gradient is chosen. The value of the gradient along x_i is given by:

$$\frac{\partial gini^D(S, a \leq v_l)}{\partial x_i} = \frac{2}{n_l(n - n_l)}(c_i\frac{n_l}{n} - x_i) - \frac{1}{n}(\frac{1}{(n - n_l)^2}\sum_{i=1}^{c}(c_i - x_i)^2 - \frac{1}{n_l^2}\sum_{i=1}^{c}x_i^2) \quad (2)$$

We calculate $gini^{est}$ for interval $[v_l, v_u]$ from left to right as well as right to left. Let class j give the minimum gradient at v_l. Then,

$$\frac{2}{n_l(n - n_l)}(c_j\frac{n_l}{n} - x_j) \leq \frac{2}{n_l(n - n_l)}(c_i\frac{n_l}{n} - x_i) \quad \forall\ 1 \leq i \leq c \quad (3)$$

From formula (3), we obtain :

$$c_j\frac{n_l}{n} - x_j \leq c_i\frac{n_l}{n} - x_i \quad \forall\ 1 \leq i \leq c \quad (4)$$

Thus,

$$c_j\frac{n_l + 1}{n} - (x_j + 1) \leq c_i\frac{n_l + 1}{n} - x_i \quad \forall\ 1 \leq i \leq c \quad (5)$$

Thus, the class with the minimum gradient will remain the class with the minimum gradient for the next split point. This property can be exploited to reduce the complexity of our hill-climbing based estimation algorithm by assuming that all points from the same class appear consecutively. Thus, the time requirements are proportional to the number of classes rather than the number of split points in the interval.

Our hill-climbing algorithm for determining $gini^{est}$ starts from the left boundary, and finds the class with the minimum gradient. Then, it evaluates the *gini* index at a new splitting point, which is at the old splitting point (left boundary) plus the number of elements in the interval with class with the minimum gradient. In case the new *gini* value is less than the current $gini^{est}$, it becomes the new $gini^{est}$ and the process is repeated for the remaining classes (and smaller interval). The same process is applied from right to left.

The *SSE* method can be modified to further prune the intervals (from the *alive* intervals) as we update

the global minimum *gini* based on computation of *alive* intervals processed at a given point. Further, it can prioritize the evaluation of the *alive* intervals such that we start with the intervals with the least estimated *gini* value. These optimizations have been incorporated in the SSE method.

The performance of the SSE method depends on the quality of the estimated lower bound of the *gini* index, and on the fraction of elements in the *alive* intervals; this fraction is given by the *survival* ratio. Since the *gini* index is calculated for each of the splitting points in these intervals, the computational cost of this phase is directly proportional to the *survival* ratio.

The SS method is simpler to implement and it requires less computational and I/O time than the SSE method for determining the splitter. On the other hand, the SSE method should generate more accurate splitters with respect to the *gini* index. The trade-off between quality and computational and I/O requirements are described in the next section.

Experimental Results We experimentally evaluated the three methods for a number of real and synthetic datasets. Table 1 gives the characteristics of these datasets. The first four datasets are taken from STATLOG projects (a widely used benchmark in classification).[1] The "Abalone", "Waveform" and "Isolet" datasets can be found in (Murphy & Aha 1994). The "Synth1" and "Synth2" datasets have been used in (Agrawal, Mehta, & Rissanen 1996; Agrawal, Mehta, & Shafer 1996) for evaluating SLIQ and SPRINT; they have been referred as "Function2" dataset.

The main parameter of our algorithm is the number of intervals used along each of the numeric attributes. We studied the following characteristics of our algorithms for different number of intervals: (1) We compare the value of *gini* index of the splitters generated by the SS and SSE methods to the *gini* index of the splitting point using DS and DM. (2) For SSE, we obtained the percentage of intervals for which the estimated value was higher than the actual value. These are measured by the number of *misses* by our method. For the missed intervals, we also measured the maximum and the average difference in the estimated value and the actual minimum for that interval. Further, we obtained the fractional number of elements that survived for the next pass (*survival* ratio).

Table 2 shows the results of the estimated *gini* index using the DS method. We used datasets for three random samples chosen from the dataset and have reported the minimum and the maximum *gini* index obtained. These results show that the DS method, in the worst case, may obtain splitters with relatively larger values of *gini* index as compared to a direct calculation. This may be partly due to the small size of the datasets. One may expect better accuracy for large datasets. Our results described in the next section show that this can have a negative impact on the overall classification error.

Table 3 shows the estimated *gini* index using the SS method and the DM method. These results show that the SS method missed the *exact gini* index most of the times for the test datasets; though the difference between the exact and estimated *gini* was not substantial. The "Letter" dataset has 16 distinct values along each of the numeric attributes. The SS method, with number of intervals greater than 16, should work well for such a dataset. The same holds true for the "Satimage" and "Shuttle" datasets as they have a small number of distinct values.

The results in Table 4 show the estimated *gini* index using the SSE method and the DM method. These results show that the SSE method missed the *exact gini* index only for a few cases. Further, the estimated *gini* generated by the SSE method is more accurate than those of the SS method (for the missed ones). Tables 3 and 4 show that the accuracy of the SS method is more sensitive to the number of intervals which makes the SSE method more scalable. This experiment shows that the SSE method is a definite improvement, more scalable, and robust than the SS method.

Our experimental results (not reported here) also show that the number of misses for the SSE method are very small. Further, for the missed intervals, the difference between the actual minimum and estimated minimum was extremely small (Alsabti, Ranka, & Singh 1998).

The results in Table 5 show that the *survival* ratio of the SSE method is less than 10% for all the test datasets for more than 10 intervals, and less than a few percent for more than 100 intervals. Further, the *survival* ratio generally decreased with increase in the number of intervals.

The above results clearly demonstrate the effectiveness of the SSE method; it can achieve a very good estimate of the splitter with minimum *gini* index and require limited amount of memory. Further, it has a better performance than algorithms that utilize only a small subset of the overall sample for obtaining the splitters.

For each point in an *alive* interval the class information, as well as the value of the numeric attribute corresponding to the *alive* interval, needs to be stored and processed. Assuming that all categorical and numeric attributes as well as class field are of the same size (in bytes), the amount of data to be processed for all the *alive* intervals is proportional to $O(2 \times SurvivalRatio \times NumberOfNumericAttributes \times NumberofRecords)$. This corresponds to $(2 \times SurvivalRatio \times \frac{NumberOfNumericAttributes}{NumberOfAllAttributes+1})$ fraction of the original dataset.

We use quantiling-based technique to generate the intervals. Hence, the domain sizes of the numeric attributes do not have any impact on the performance

[1] For more details about the STATLOG datasets, the reader is referred to http://www.kdnuggets.com/.

Dataset	No of examples	No. of attributes	No. of numeric attributes	No. of classes
Letter	20,000	16	16	26
Satimage	6,435	36	36	6
Segment	2,310	19	19	7
Shuttle	58,000	9	9	7
Abalone	4,177	8	7	29
Waveform-21	5,000	21	21	3
Waveform-40	5,000	40	40	3
Isolet	7,797	617	617	26
Synth1	400,000	9	6	2
Synth2	800,000	9	6	2

Table 1: Description of the datasets

Dataset	Exact *gini*	10%		15%		20%	
		Min	Max	Min	Max	Min	Max
Letter	0.940323	0.940323	**0.942294**	0.940323	**0.942326**	0.940323	**0.942326**
Satimage	0.653167	0.653167	**0.659033**	0.653167	**0.659033**	0.653167	**0.659033**
Segment	0.714286	0.714286	**0.717774**	0.714286	**0.714788**	0.714286	**0.721099**
Shuttle	0.175777	0.175777	0.175777	0.175777	0.175777	0.175777	0.175777
Abalone	0.862017	**0.862579**	**0.866956**	**0.862579**	**0.867807**	**0.862114**	**0.866992**
Waveform-21	0.546150	**0.547439**	**0.552724**	**0.547083**	**0.548120**	**0.548144**	**0.550018**
Waveform-40	0.541134	**0.548980**	**0.554132**	**0.543133**	**0.554132**	**0.542480**	**0.550520**
Isolet	0.926344	**0.926552**	**0.932994**	0.926344	**0.926883**	0.926344	**0.927038**

Table 2: Exact and estimated *gini* using DS for different sampling ratios (3 runs)

Dataset	Exact *gini*	Number of Intervals					
		200	100	50	25	10	5
Letter	0.940323	0.940323	0.940323	0.940323	0.940323	**0.941751**	**0.941751**
Satimage	0.653167	0.653167	0.653167	**0.654601**	**0.654601**	**0.654601**	**0.682317**
Segment	0.714286	0.714286	0.714286	**0.717774**	0.714286	**0.721154**	**0.740018**
Shuttle	0.175777	0.175777	0.175777	0.175777	0.175777	**0.181001**	**0.238166**
Abalone	0.862017	**0.862477**	**0.862697**	**0.862752**	**0.862907**	**0.862724**	**0.865768**
Waveform-21	0.546150	**0.547003**	**0.547464**	**0.547353**	**0.547424**	**0.548090**	**0.550608**
Waveform-40	0.541134	**0.541680**	**0.541943**	**0.542257**	**0.542894**	**0.543224**	**0.543224**
Isolet	0.926344	**0.927403**	**0.927290**	**0.927424**	**0.927403**	**0.929721**	**0.930125**

Table 3: Exact minimum and estimated *gini* based on SS using different number of intervals

Dataset	Exact *gini*	Number of Intervals					
		200	100	50	25	10	5
Letter	0.940323	0.940323	0.940323	0.940323	0.940323	**0.941751**	0.940323
Satimage	0.653167	0.653167	0.653167	0.653167	0.653167	0.653167	0.653167
Segment	0.714286	0.714286	0.714286	0.714286	0.714286	**0.715799**	**0.721099**
Shuttle	0.175777	0.175777	0.175777	0.175777	0.175777	0.175777	0.175777
Abalone	0.862017	0.862017	0.862017	0.862017	0.862017	0.862017	0.862017
Waveform-21	0.546150	0.546150	0.546150	0.546150	0.546150	0.546150	0.546150
Waveform-40	0.541134	0.541134	0.541134	0.541134	0.541134	0.541134	0.541134
Isolet	0.926344	**0.926807**	**0.926807**	0.926344	**0.927138**	0.926344	0.926344

Table 4: Exact minimum and estimated *gini* based on SSE using different number of intervals

Dataset	Number of Intervals					
	200	100	50	25	10	5
Letter	0.000	0.000	0.000	0.002	0.029	0.174
Satimage	0.000	0.000	0.001	0.006	0.022	0.092
Segment	0.000	0.000	0.003	0.000	0.011	0.011
Shuttle	0.000	0.000	0.000	0.003	0.001	0.032
Abalone	0.005	0.009	0.027	0.085	0.361	0.436
Waveform-21	0.007	0.012	0.022	0.036	0.143	0.347
Waveform-40	0.001	0.002	0.005	0.014	0.062	0.175
Isolet	0.000	0.000	0.000	0.001	0.002	0.019

Table 5: The fractional *survival* ratio using different number of intervals using the SSE method

of our algorithm. The memory requirements is proportional to $q \times (c + f)$, where q is the number of intervals, c is the number of classes, and f is the number of numeric attributes. The conducted experiments showed that 50-200 intervals are sufficient to achieve an acceptable performance. The amount of storage required for keeping the relevant information for these intervals is much smaller than storing the whole data set in the main memory. We expect this to be the case for larger datasets; though further investigation is required.

The CLOUDS Algorithm

In this section we describe our classifier, which uses a breadth-first strategy to build the decision tree and uses the *gini* index for evaluating the split points. It uses either the SS method or SSE method to derive the splitter at each node of the tree. CLOUDS evaluates the split points for categorical attributes as in SPRINT. However, the evaluation of split points for numeric attributes is different (as described below). We also describe the partitioning step for decomposing an internal node.

Numeric Attribute For each node at level l of the tree, the range of each numeric attribute is partitioned into q intervals. A random sample S of size s is derived from the dataset. This step requires one read operation of the whole dataset, which is needed only at the root level. The samples for the lower levels are obtained using the sample for the higher levels; $q - 1$ points are derived from S by regular sampling. This corresponds to q regularly spaced quantiles of the set S. If q is much smaller than S, we expect that the q intervals have close to $\frac{n}{q}$ records with high probability, where n is the total number of records.[2]

The dataset is read and the class frequencies are computed for each of the q intervals. At this stage, the splitter is found using either the SS or SSE methods.

For the *alive* intervals along each numeric attribute (using the SSE method), the class frequencies for intervals are computed. We assume that the size of each of the *alive* intervals is small enough that it can fit in the main memory.[3] Since we are choosing the intervals based on regular sampling of a random subset, the number of records from the whole dataset in each interval are close to $\frac{n}{q}$ with a high probability. Thus, each of these intervals will fit the memory with high probability when q is $O(\frac{n}{M})$, where M is the size of the main memory. For each *alive* interval, we sort all the points. Since we know the class frequencies at interval boundaries and the points of a *alive* intervals are sorted, the class frequencies at each point in the *alive* interval can be computed easily. For each distinct point, we evaluate the *gini* index and keep track of the minimum value, which will be used to determine the best splitting criterion.

The above operation requires reading the entire dataset corresponding to the internal node and storing only the points that belong to *alive* intervals. If all the *alive* intervals do not fit in the main memory, they are written to the disk. Another pass is required to read these *alive* intervals one at a time. If the *survival* ratio is small, and all the *alive* intervals can fit in the main memory, this extra pass is not required.

Partitioning the Dataset The splitting criterion is applied on each record to determine its partition (left or right). The number of records read and written is equal to the number of records represented by the internal node. For node i at level l of the tree, set S is partitioned, using the splitting criteria, into two sample sets S_1 and S_2. The sample sets S_1 and S_2 are used for left and right subtrees of node i, respectively. While reading the data for splitting a given node, the class frequencies for the interval boundaries are updated. Thus, frequency vectors corresponding to S1 and S2 are updated, which avoids a separate pass over the entire data to evaluate these frequencies.

Assuming that all the *alive* intervals fit in the main memory, we need to read all the records at most twice and write all the records for each internal node (except the root node). The records read and written correspond only to the internal node(s) being split. An extra

[2] Our algorithm does not require intervals to be of equal size though.

[3] Otherwise, the processing of the *alive* intervals needs to be staged.

read is required on the root to create the sample list.

We expect that CLOUDS (with the SSE method) and the SPRINT algorithm will build the same tree for most datasets, which is demonstrated by the results obtained in a later subsection. Our theoretical analysis (not reported here) shows that CLOUDS is superior to SPRINT in terms of I/O and computational requirements (Alsabti, Ranka, & Singh 1998).

Experimental Results We conducted several experiments to measure the accuracy, compactness, and efficiency of the CLOUDS classifier and compared it with SPRINT, IND-C4, IND-CART, and the DM method. In this section we briefly describe some of these results. For more details the reader is referred to a detailed version of this paper (Alsabti, Ranka, & Singh 1998).

Table 6 gives the average accuracy and the tree size of the classifiers generated by the DS method. These results and observations by other researchers (Catlett 1991; Chan & Stolfo 1993) show that one can potentially lose a significant amount of accuracy if the whole dataset is not used for building the decision tree. However, we will like to note that it may be a manifestation of the size of the dataset. It is not clear whether this will be the case for larger datasets.

We wanted to study the quality of results obtained by CLOUDS based on the size of the main memory. Our current implementation recursively decomposes the number of intervals into two subsets using a weighted-assignment strategy. The number of intervals assigned to each of the datasets is proportional to the size of the dataset. Since the size of the datasets available is reasonably small, we achieve the effects of having limited memory by using a stopping criterion based on the number of intervals assigned to a given data subset. The DM method is used when the stopping criterion is met.

We conducted a set of experiments to evaluate the SS and SSE methods based on the overall results in CLOUDS for pruned-tree size, accuracy, and execution time (see (Alsabti, Ranka, & Singh 1998) for more details). The pruned-tree sizes generated by the SS method (not shown here) are comparable to those of the SSE method. However, the SSE method achieved slightly better accuracy for most of the cases (comparison not shown here). The SSE method requires at most 30% more time than the SS algorithm (see Table 8). These results show that the overhead of using SSE over SS is not very large. Given that the SSE method is more scalable and less sensitive to the number of intervals, it may be the better option for most datasets. The results presented in the rest of this section assume that SSE method was used.

Table 7 gives the quality of decision trees in terms of accuracy and size (after FULL MDL-based pruning (Agrawal, Mehta, & Rissanen 1996; 1995)) using CLOUDS for different number of intervals at the root. These assume that the DM method was used when the number of intervals was less than 10. The corresponding results for CART, C4, SPRINT and the DM method are also presented.[4] The results for the CART, C4, and SPRINT algorithms have been derived from the literature (Agrawal, Mehta, & Rissanen 1996; Agrawal, Mehta, & Shafer 1996). These results show that the quality of results obtained is similar or comparable to those of CART, C4, SPRINT and DM method both in terms of accuracy and the size of the decision tree obtained. Assuming that we started with 200 intervals and divided the datasets into approximately equal datasets at each node, a stopping criterion based on 10 intervals will correspond to application of the heuristic method for around 4 levels.

Table 8 gives the time requirements of our algorithm assuming that the DM method is applied when the size of the data is small enough that less than 10 intervals are assigned to it. These timings have been obtained from the IBM RS/6000 Model 590 workstation running AIX 4.1.4. These results assume that the *survival* ratios are small enough that the *alive* intervals can be processed in the main memory. The maximum difference between using SS versus SSE is less than 30%. Thus, the overhead of processing the *alive* intervals in SSE is not very high.

SPRINT spent around 1,000 and 2,000 seconds in classifying Synth1 and Synth2 datasets, respectively (Agrawal, Mehta, & Shafer 1996). These timings may not be directly comparable due to somewhat different experimental setups.

Dataset	SSE		SS	
	$q = 200$	$q = 100$	$q = 200$	$q = 100$
Letter	8.76	8.30	8.43	8.26
Satimage	5.40	4.57	5.43	4.60
Segment	1.06	0.9	0.97	0.87
Shuttle	3.98	3.44	3.97	3.41
Abalone	2.89	2.52	2.40	2.25
Waveform-21	4.86	4.48	4.34	4.27
Waveform-40	9.07	8.69	8.53	8.26
Isolet	377.63	331.14	289.13	270.72
Synth1	35.74	34.86	35.43	34.76
Synth2	73.35	72.06	74.09	71.79

Table 8: Time requirements (in seconds) of CLOUDS using SS and SSE. DM was applied for nodes with sizes roughly $\frac{10}{q}$ of original dataset

Conclusion

There are several options for developing efficient algorithms for performing such data mining operations on large datasets. One option is to develop algorithms that reduce computational and I/O requirements, but perform essentially the same operations. Another option is

[4]The DM and SPRINT should generate the same results. However, they generated different trees because (probably) of different implementation choices.

Dataset	5%		20%		40%		100%	
	Accuracy	Tree size	Accuracy	Tree size	Accuracy	Tree size	Accuracy	Tree size
Letter	53.6	115.0	69.1	335.0	76.7	592.3	**83.4**	**881**
Satimage	75.9	17.0	81.7	45.7	83.6	69.7	**86.4**	**127**
Shuttle	99.5	9.0	99.9	21.0	99.9	25.7	**99.9**	**27**
Abalone	24.1	15.0	24.0	39.0	24.2	85.7	**26.3**	**137**
Isolet	55.6	57.7	73.1	107.0	78.0	169.0	**82.9**	**261**

Table 6: Average (of three trials) pruned tree size and accuracy using the *DS* method

Dataset	IND-CART	IND-C4	SPRINT	*DM*	CLOUDS with *SSE*	
					$q = 200$	$q = 100$
Letter	84.7/1199.5	86.8/3241.3	84.6/1141	83.4/881	83.3/893	83.4/881
Satimage	85.3/90	85.2/563	86.3/ 159	86.4/127	85.9/135	84.9/111
Segment	94.9/ 52	95.9/102	94.6/18.6	94.4/41.0	94.7/55.2	95.0/51.2
Shuttle	99.9/27	99.9/57	99.9/29	99.9/27	99.9/41	99.9/33
Abalone	–	–	–	26.3/137	26.4/147	25.9/135
Waveform-21	–	–	–	77.3/168.4	76.8/172.8	77.7/172.8
Waveform-40	–	–	–	76.4/187.4	76.0/189.8	77.2/184.6
Isolet	–	–	–	82.9/261	82.6/ 255	82.7/255

Table 7: The accuracy/pruned-tree size obtained for different datasets. The *DM* method was applied in CLOUDS for nodes with dataset sizes equal to (roughly) $\frac{10}{q}$ of the original dataset

to develop algorithms which reduce the computational and I/O requirements by performing an approximation of the actual operation without significant or no loss of accuracy. The CLOUDS classifier falls in the latter category, while SPRINT falls in the former category. However, CLOUDS has been shown to be as accurate as SPRINT and shown to have substantially superior computational characteristics. We believe that such an approach may provide attractive alternatives to direct methods for other data mining applications especially for cases in which an implicit optimization needs to be performed.

It will be interesting to derive heuristics that will guarantee that the estimate is always lower than the actual value. However, this value should be close enough to the actual minimum possible for the *survival* ratio to be small. The new algorithms can potentially be extended to other splitting functions, e.g., the towing function and the gain ratio.

Acknowledgments

The work of Khaled Alsabti was done while he was visiting the department of CISE at University of Florida. The work of Sanjay Ranka was supported in part by AFMC and ARPA under F19628-94-C-0057 and WM-82738-K-19 (subcontract from Syracuse University) and in part by ARO under DAAG 55-97-1-0368 and Q000302 (subcontract from NMSU). The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

Agrawal, R.; Mehta, M.; and Rissanen, J. 1995. MDL-Based Decision Tree Pruning. *Int'l Conf. on Knowledge Discovery in Databases and Data Mining* 216–221.

Agrawal, R.; Mehta, M.; and Rissanen, J. 1996. SLIQ: A Fast Scalable Classifier for Data Mining. *Proc. of the Fifth Int'l Conference on Extending Database Technology* 18–32.

Agrawal, R.; Mehta, M.; and Shafer, J. C. 1996. SPRINT: A Scalable Parallel Classifier for Data Mining. *Proc. of the 22th Int'l Conference on Very Large Databases* 544–555.

Alsabti, K.; Ranka, S.; and Singh, V. 1998. CLOUDS: A Decision Tree Classifier for Large Datasets. *http://www.cise.ufl.edu/~ranka/.*

Catlett, J. 1991. *Megainduction: Machine Learning on Very Large Databases.* Ph.D. thesis, University of Sydney.

Chan, P. K., and Stolfo, S. J. 1993. Meta-Learning for Multistrategy and Parallel Learning. *Proc. Second Int'l Workshop on Multistrategy Learning* 150–165.

Murphy, P. M., and Aha, D. W. 1994. UCI Repository of Machine Learning Databases. *http://www.ics.uci.edu/ mlearn/MLRepository.html, Irvine, CA: University of California, Department of Information and Computer Science.*